Virtual Memory: Concepts

15-213: Introduction to Computer Systems
"17th" Lecture, July 9, 2019

Instructor:
Sol Boucher

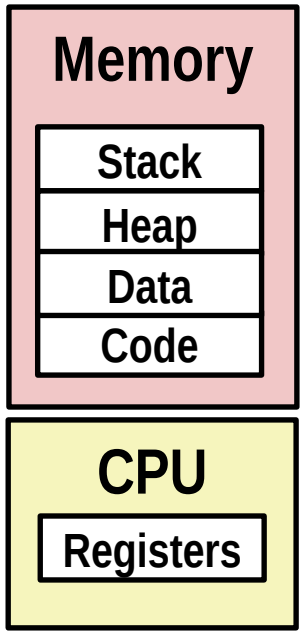
Today

- **Processes: Concepts**
- Address spaces
- VM as a tool for caching
- VM as a tool for memory management
- VM as a tool for memory protection

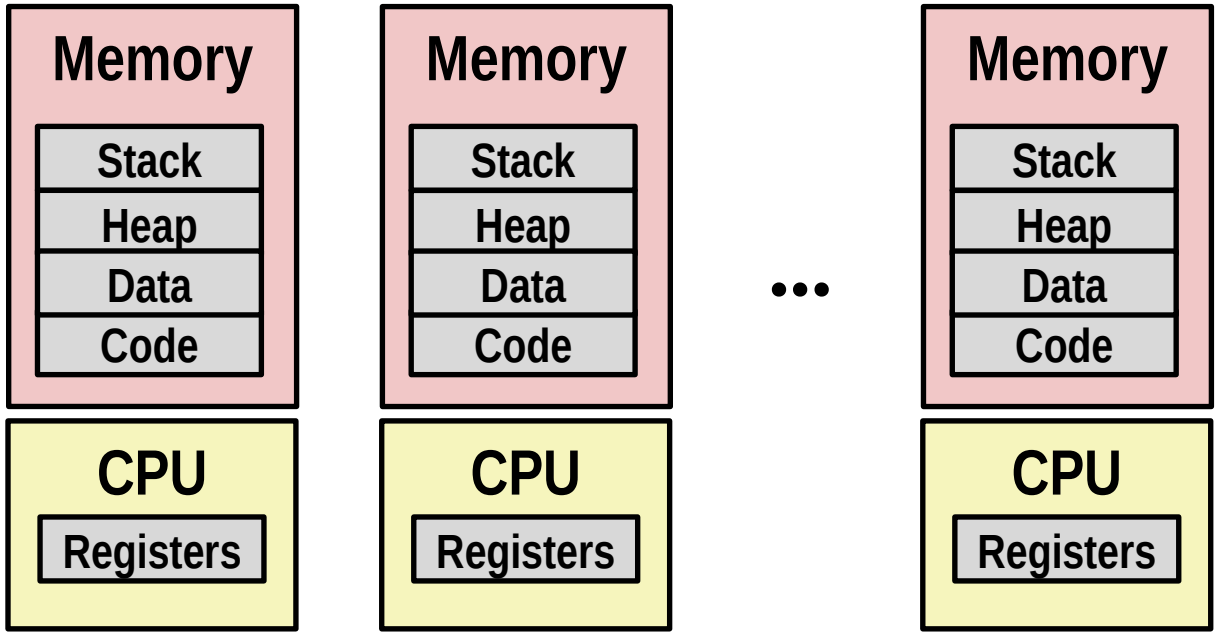
Processes

- **Definition: A *process* is an instance of a running program.**
  - One of the most profound ideas in computer science
  - Not the same as "program" or "processor"

- **Process provides each program with two key abstractions:**
  - ***Logical control flow***
    - Each program seems to have exclusive use of the CPU
    - Provided by kernel mechanism called *context switching*
  - ***Private address space***
    - Each program seems to have exclusive use of main memory.
    - Provided by kernel mechanism called *virtual memory*

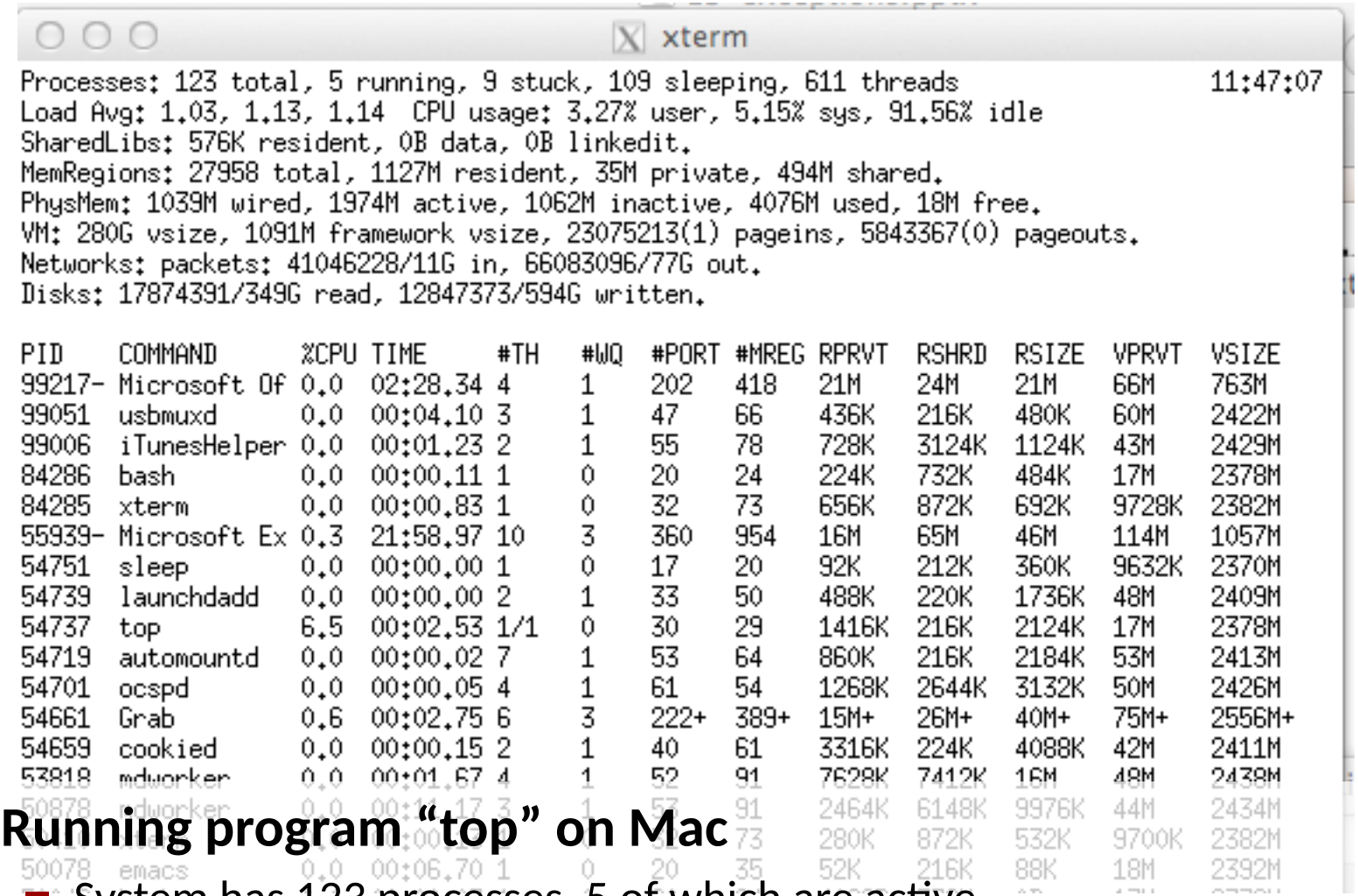


Multiprocessing: The Illusion



- **Computer runs many processes simultaneously**
  - Applications for one or more users
    - Web browsers, email clients, editors, …
  - Background tasks
    - Monitoring network & I/O devices

Multiprocessing Example



- **Running program "top" on Mac**
  - System has 123 processes, 5 of which are active
  - Identified by Process ID (PID)

Preview: Creating and Terminating Processes

From a programmer's perspective, we can think of a process as being in one of three states

- **Running**
  - Process is executing (or waiting to, as we'll see later this week)
- **Stopped**
  - Process execution is *suspended* until further notice (covered later)
- **Terminated**
  - Process is stopped permanently

Terminating Processes

- **Process becomes terminated for one of three reasons:**
  - Returning from the **`main`** routine
  - Calling the **`exit`** function
  - One other that we'll discuss next week…
- **`void exit(int status)`**
  - Terminates with an *exit status* of **`status`**
  - Convention: normal return status is 0, nonzero on error
  - Another way to explicitly set the exit status is to return an integer value from the main routine
- **`exit` is called once but never returns.**

Creating Processes

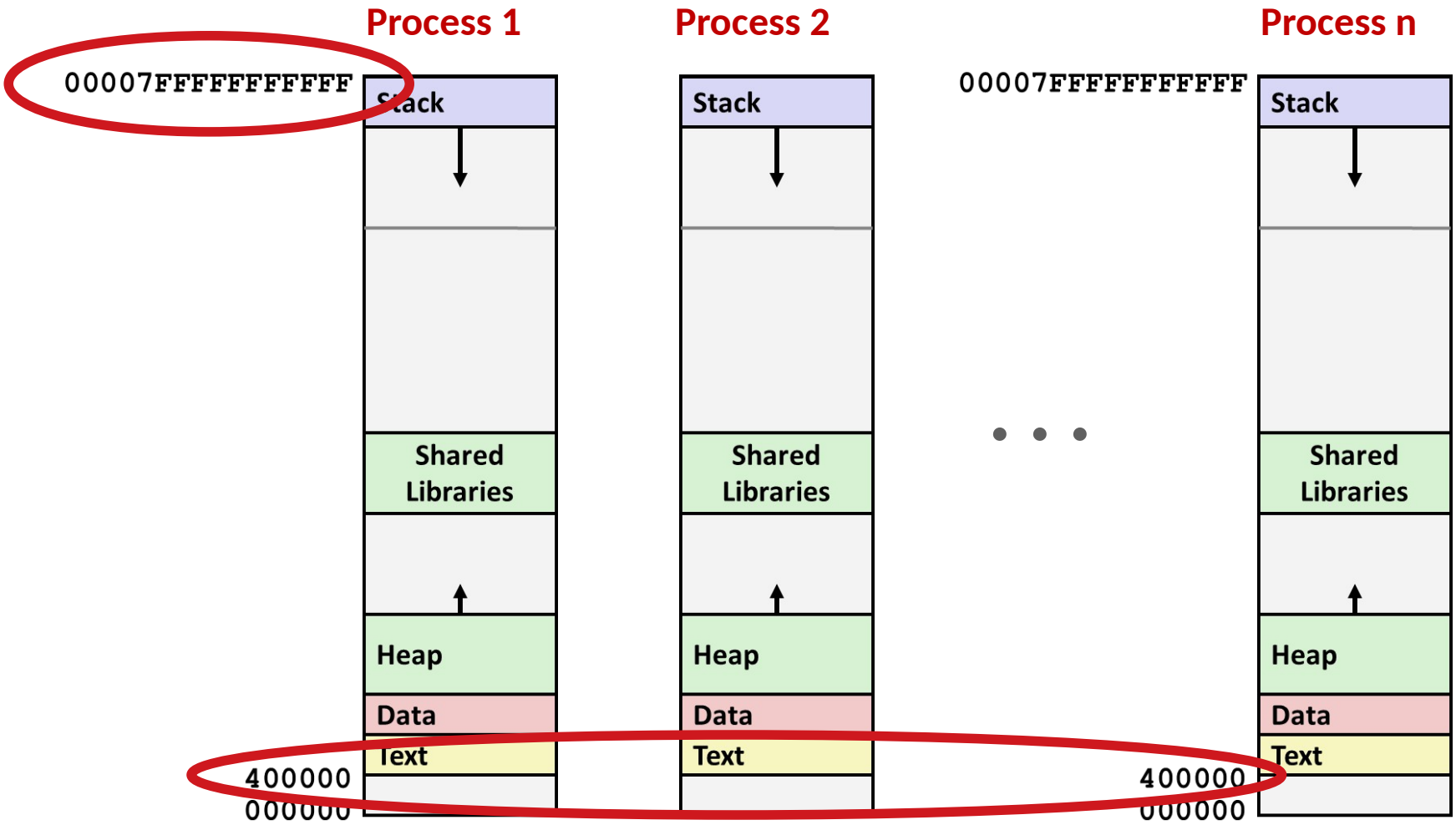
- ***Parent process*** **creates a new running** ***child process*** **by calling `fork`**

- **`int fork(void)`**
  - Returns 0 to the child process, child's PID to parent process
  - Child is *almost* identical to parent...

Activity: part 1

- **`fork` is interesting (and often confusing) because it is called *once* but returns *twice***

Hmmm, How Does This Work?!



Solution: Virtual Memory (today and next lecture)

Creating Processes

- ***Parent process*** **creates a new running** ***child process*** **by calling `fork`**

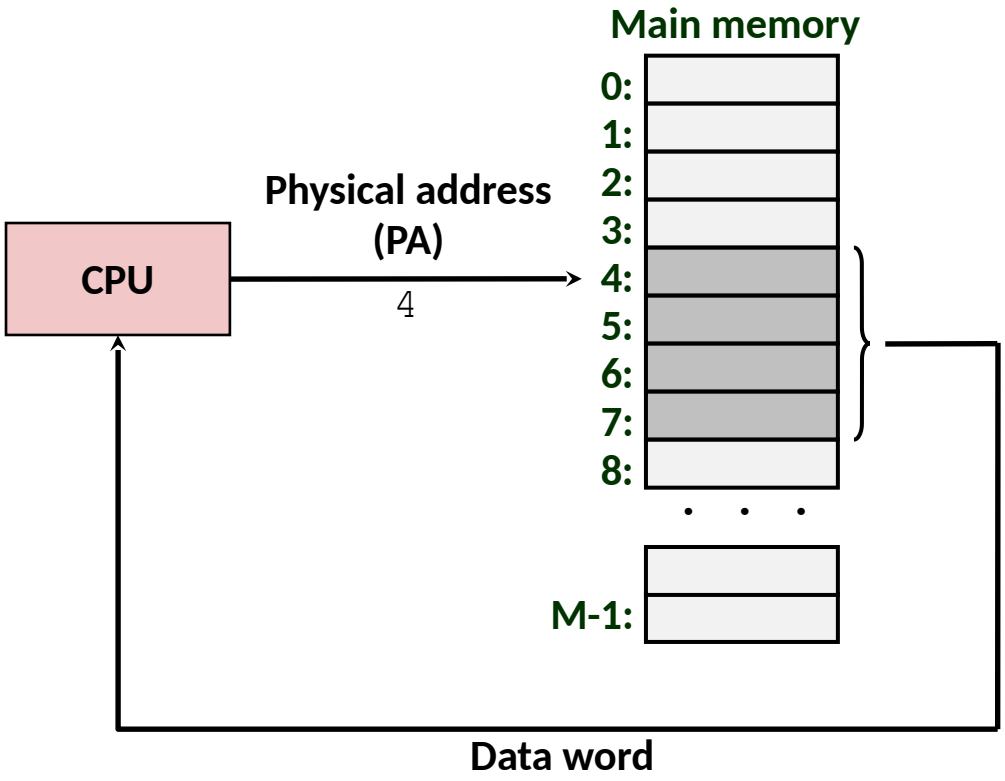
- **`int fork(void)`**
  - Returns 0 to the child process, child's PID to parent process
  - Child is *almost* identical to parent:
    - Child get an identical (but separate) copy of the parent's virtual address space.
    - Child gets identical copies of the parent's open file descriptors
    - Child has a different PID than the parent

- **`fork` is interesting (and often confusing) because it is called *once* but returns *twice***

Today

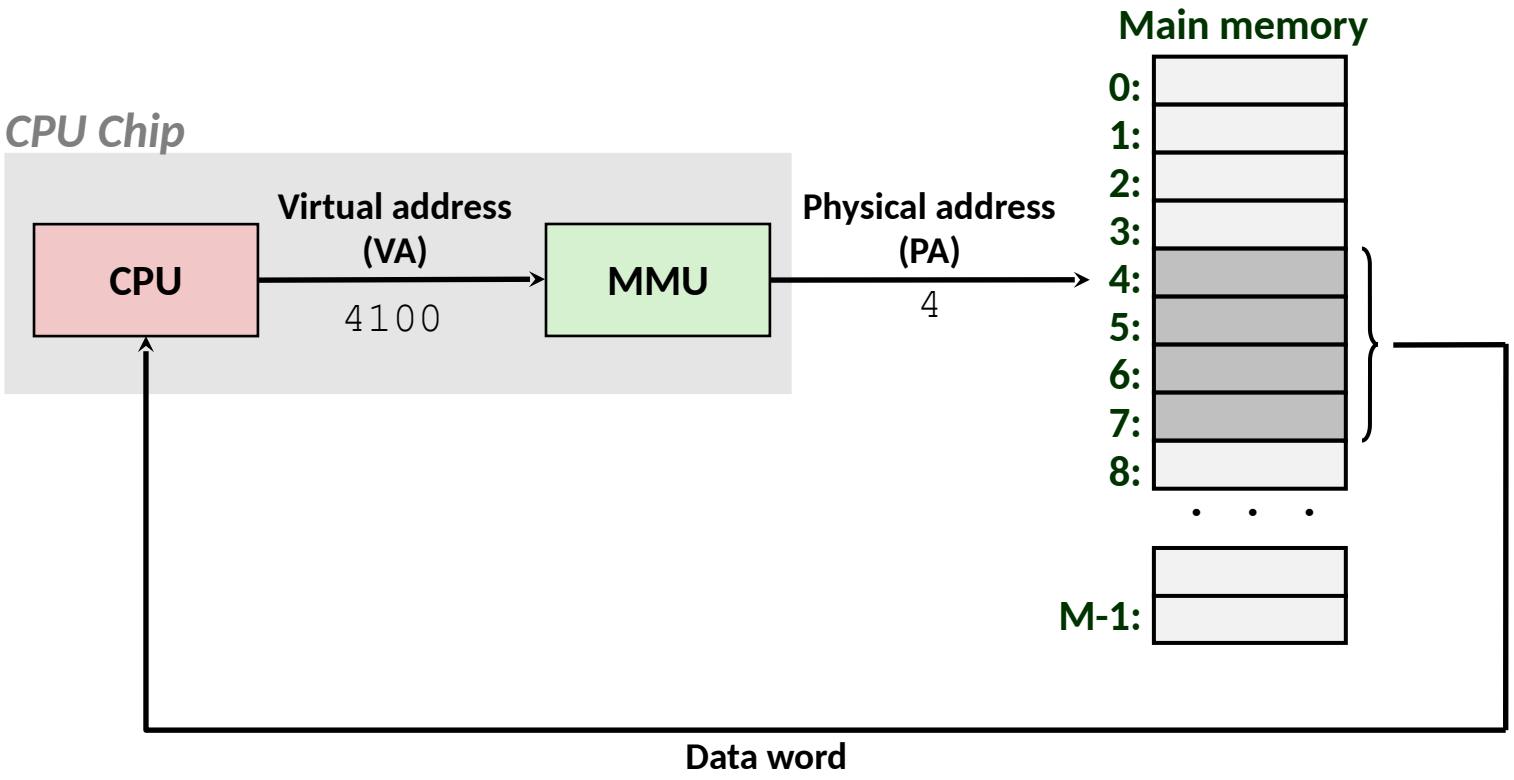
- Processes: Concepts
- **Address spaces**
- VM as a tool for caching
- VM as a tool for memory management
- VM as a tool for memory protection

A System Using Physical Addressing



- Used in "simple" systems like embedded microcontrollers in devices like cars, elevators, and digital picture frames

A System Using Virtual Addressing



- Used in all modern servers, laptops, and smart phones
- One of the great ideas in computer science

Address Spaces

- **Linear address space:** Ordered set of contiguous non-negative integer addresses:
  {0, 1, 2, 3 … }

- **Virtual address space:** Set of N = 2^n virtual addresses
  {0, 1, 2, 3, …, N-1}

- **Physical address space:** Set of M = 2^m physical addresses
  {0, 1, 2, 3, …, M-1}

Why Virtual Memory (VM)?

- **Uses main memory efficiently**
  - Use DRAM as a cache for parts of a virtual address space
- **Simplifies memory management**
  - Each process gets the same uniform linear address space
- **Isolates address spaces**
  - One process can't interfere with another's memory
  - User program cannot access privileged kernel information and code

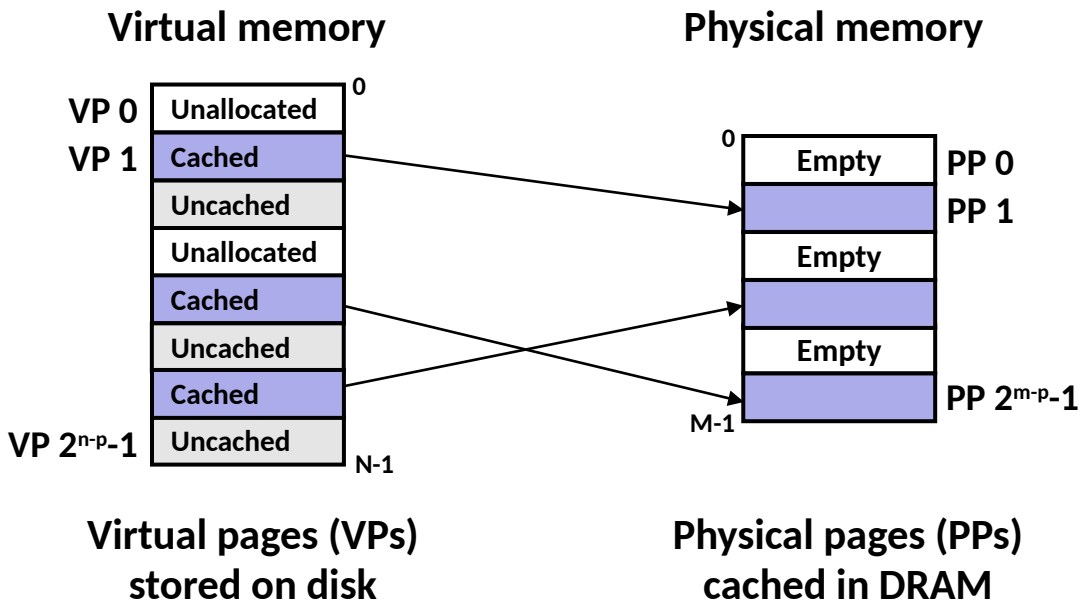
Activity: parts 2 and 3

Today

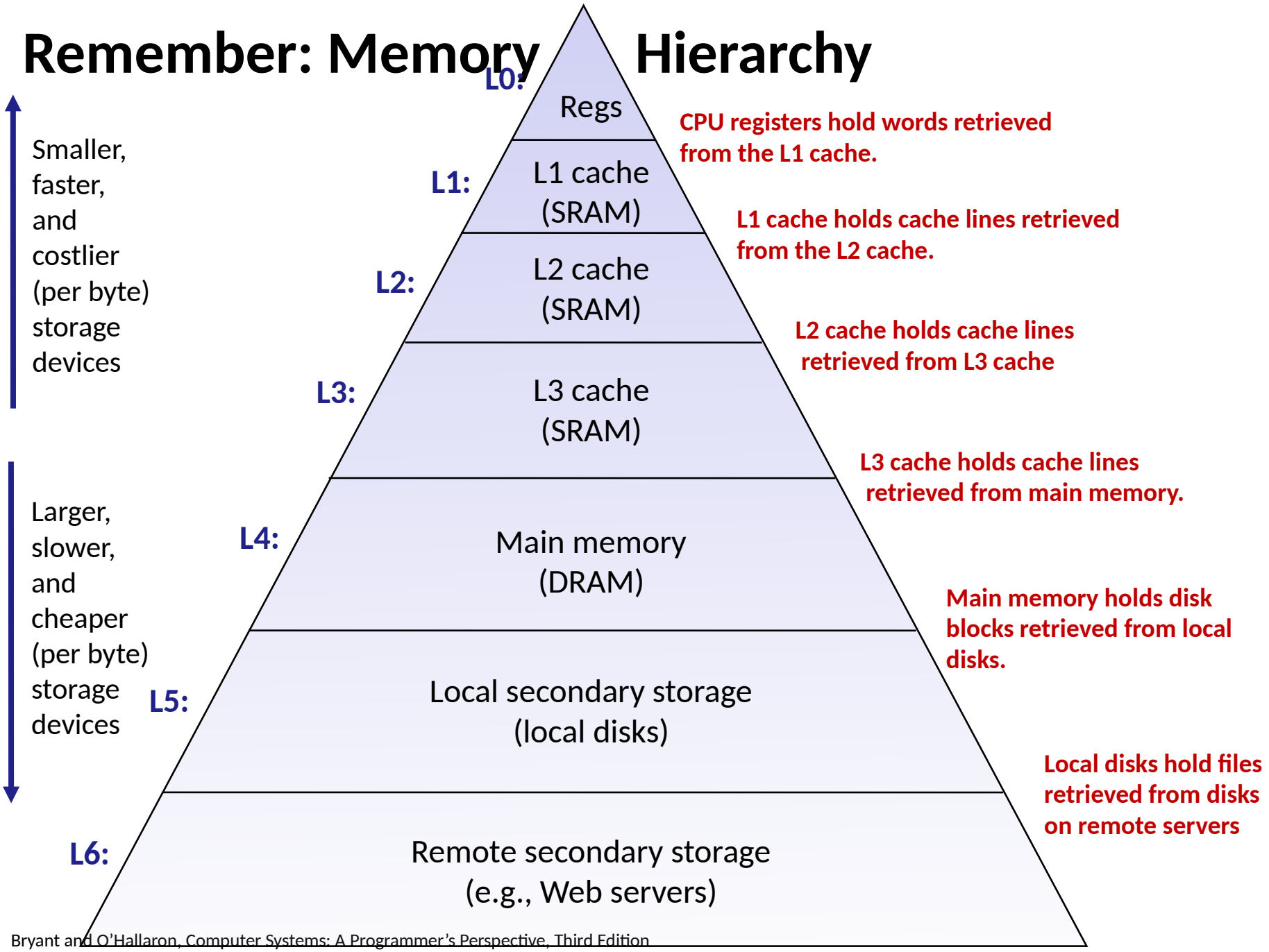
- Processes: Concepts
- Address spaces
- **VM as a tool for caching**
- VM as a tool for memory management
- VM as a tool for memory protection

VM as a Tool for Caching

- Conceptually, *virtual memory* is an array of N contiguous bytes stored on disk.
- The contents of the array on disk are cached in *physical memory* (*DRAM cache*)
  - These cache blocks are called *pages* (size is P = 2^p bytes)



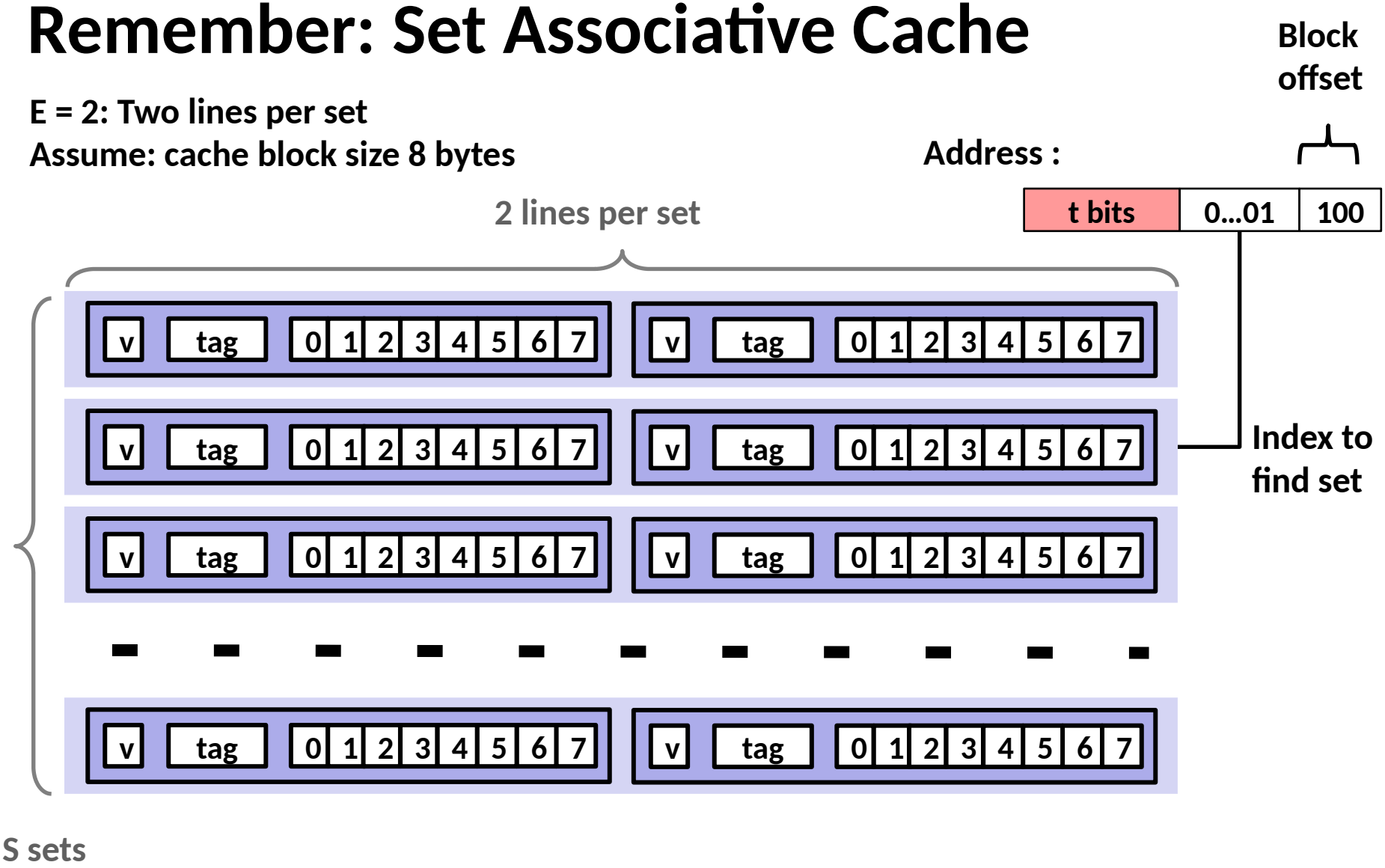
Remember: Memory Hierarchy



Remember: Set Associative Cache

E = 2: Two lines per set

Assume: cache block size 8 bytes



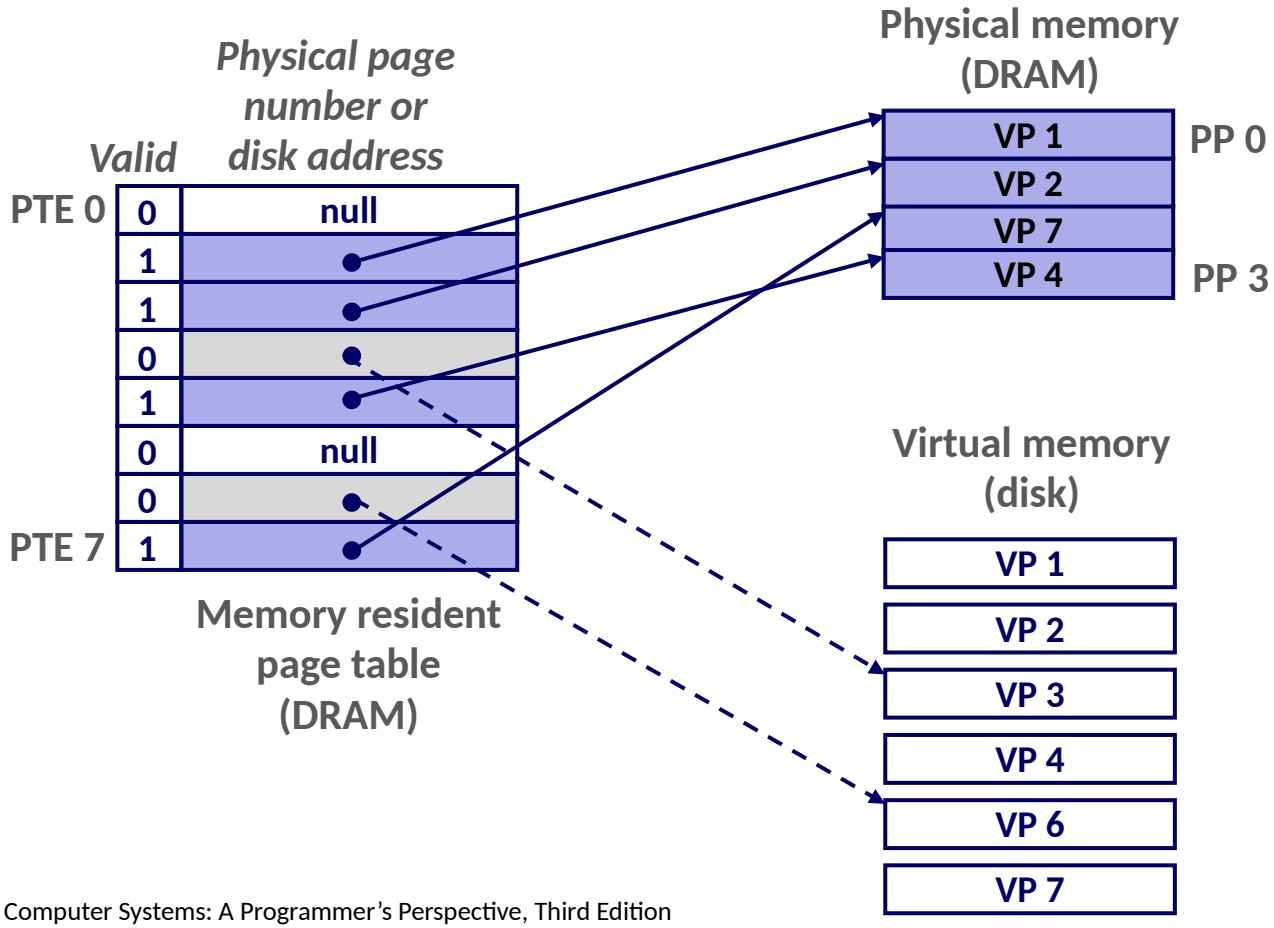
DRAM Cache Organization

- **DRAM cache organization driven by the enormous miss penalty**
  - DRAM is about ***10x*** slower than SRAM
  - Disk is about ***10,000x*** slower than DRAM

- **Consequences**
  - Large page (block) size: typically 4 KB, sometimes 4 MB
  - Fully associative
    - Any VP can be placed in any PP
    - Requires a “large” mapping function – different from cache memories
  - Highly sophisticated, expensive replacement algorithms
    - Too complicated and open-ended to be implemented in hardware
  - Write-back rather than write-through

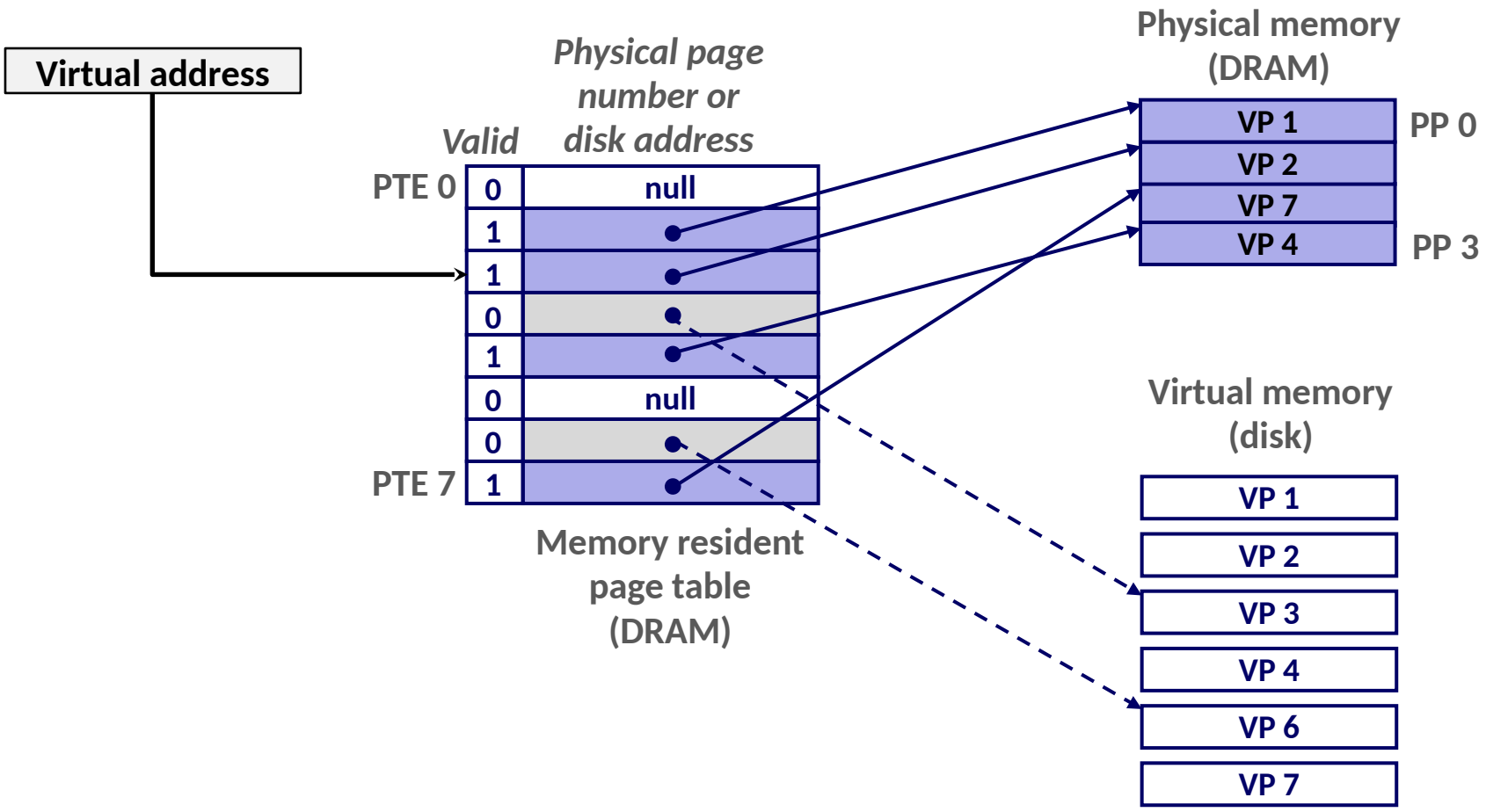
Enabling Data Structure: Page Table

- **A *page table* is an array of page table entries (PTEs) that maps virtual pages to physical pages.**
  - Per-process kernel data structure in DRAM



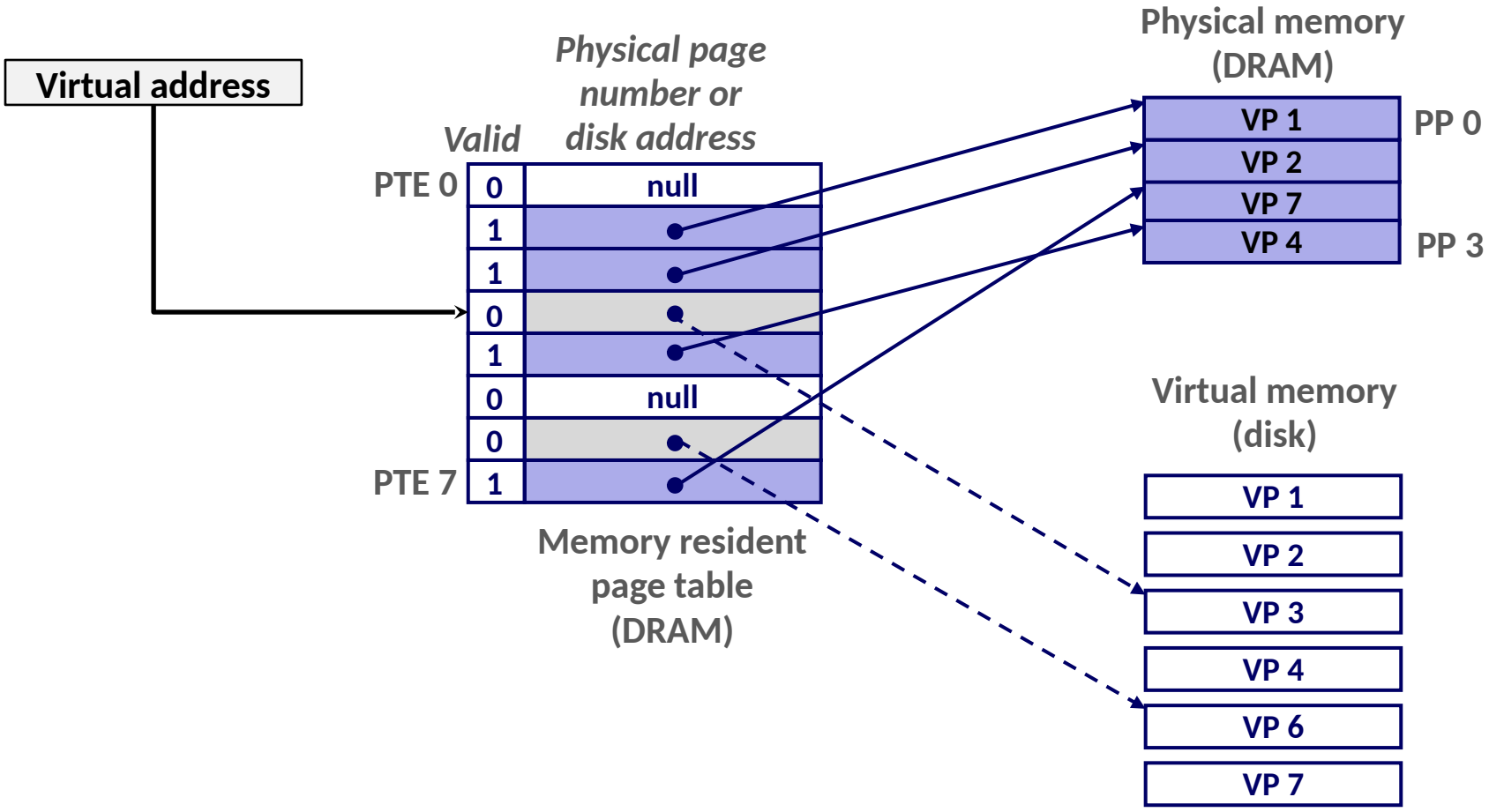
Page Hit

- ***Page hit:*** **reference to VM word that is in physical memory (DRAM cache hit)**



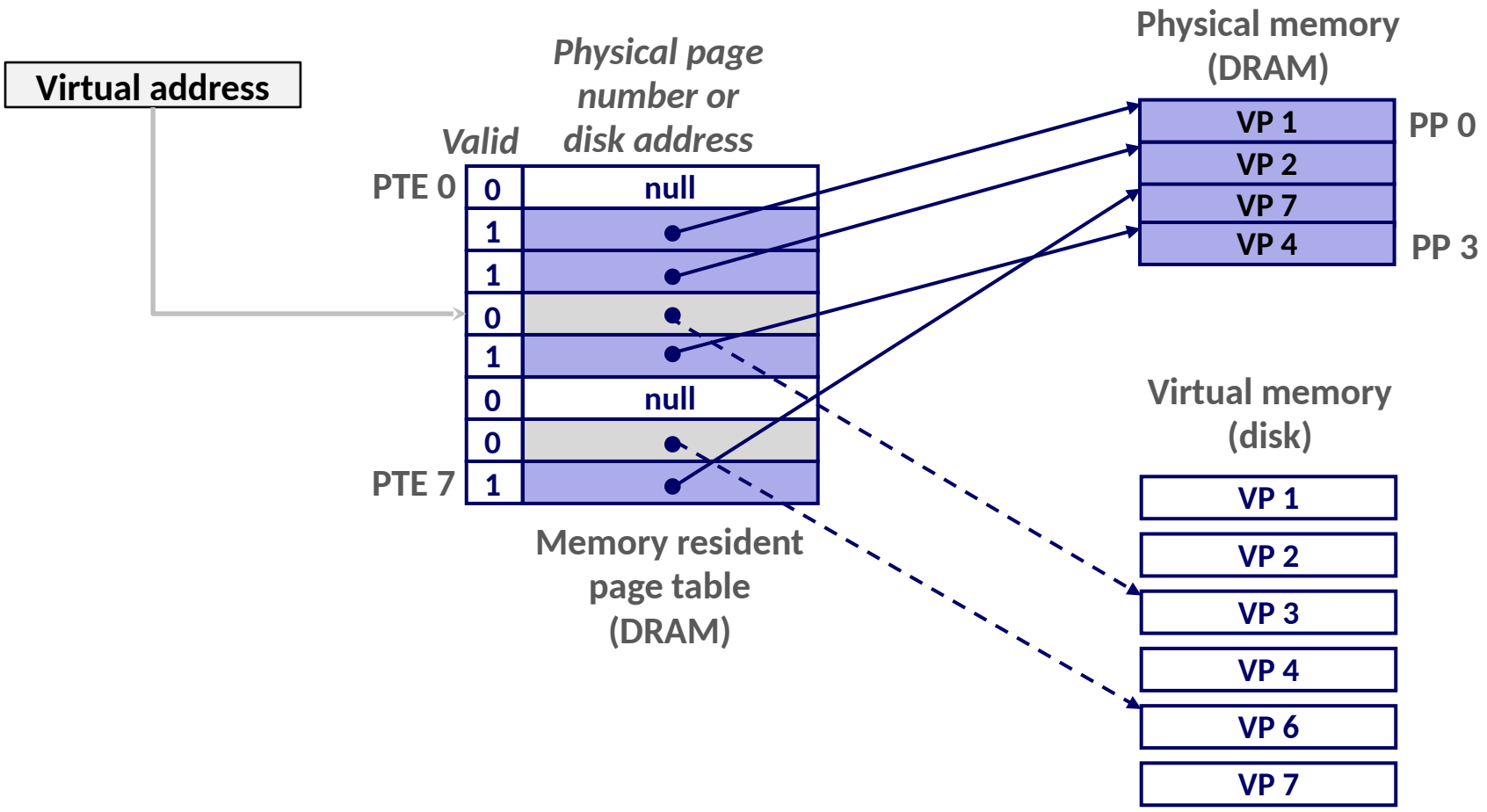
Page Fault

- ***Page fault:*** **reference to VM word that is not in physical memory (DRAM cache miss)**



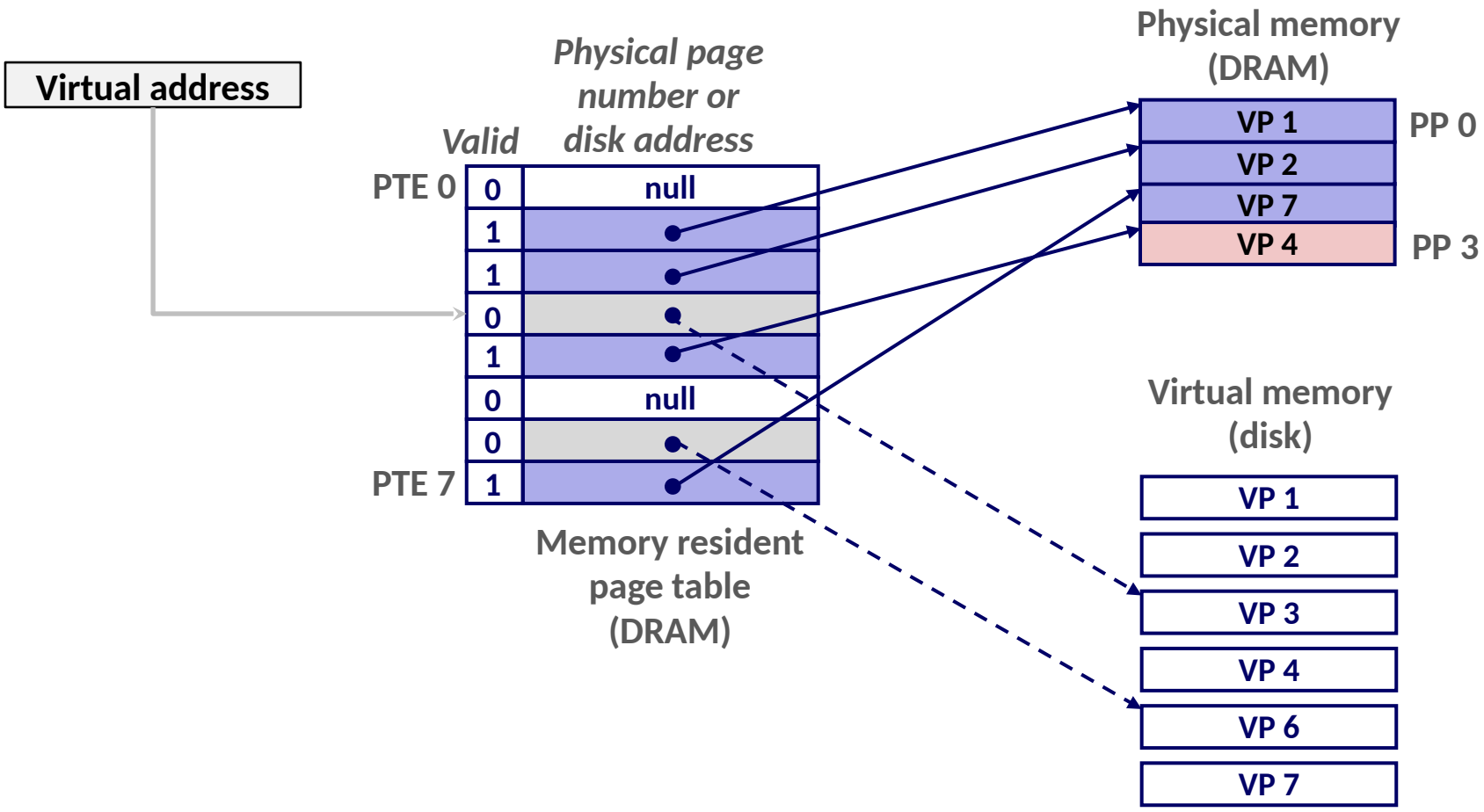
Handling Page Fault

- Page miss causes page fault (an exception)



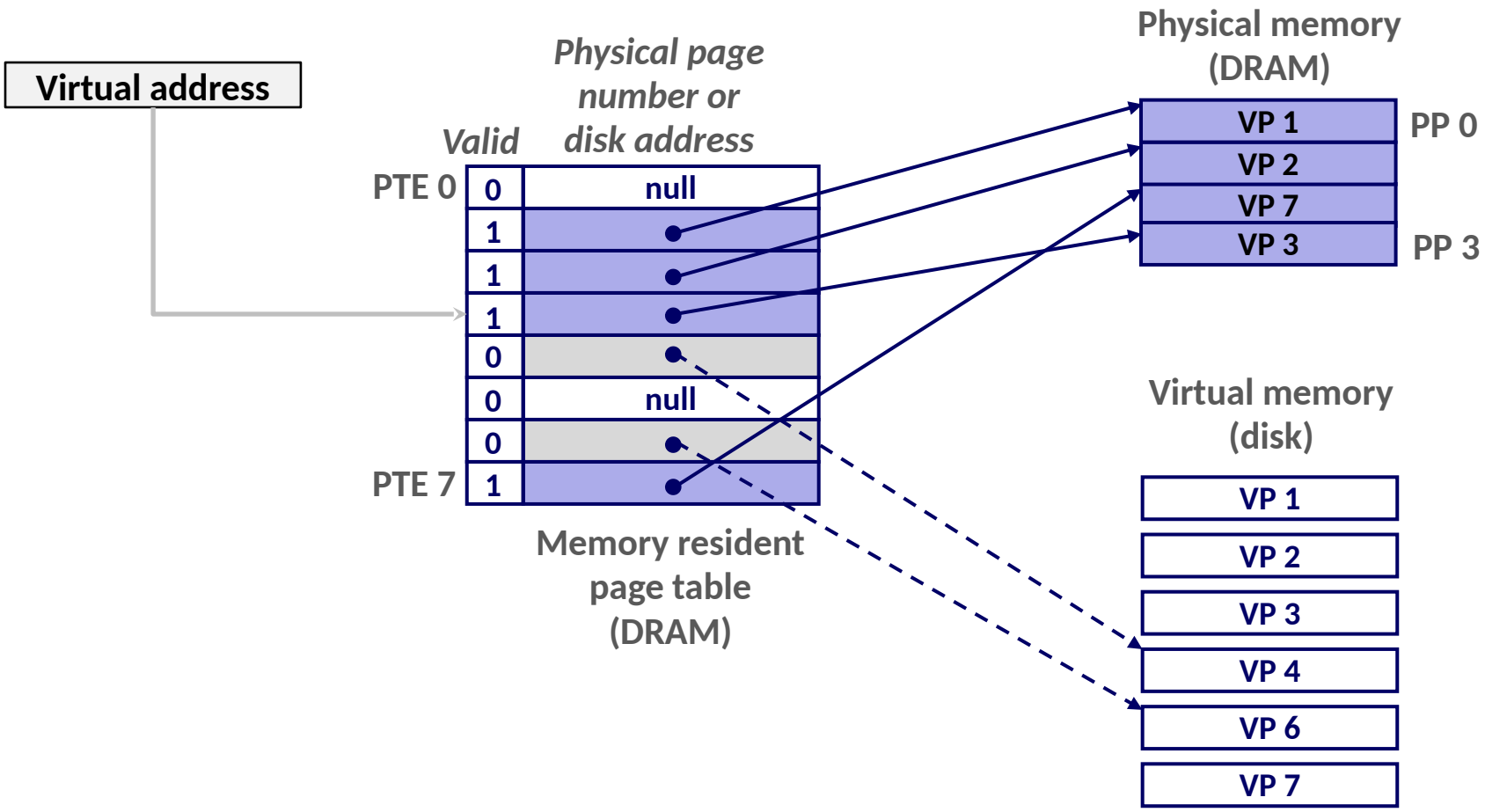
Handling Page Fault

- Page miss causes page fault (an exception)
- Operating system selects a victim to be evicted (here VP 4)



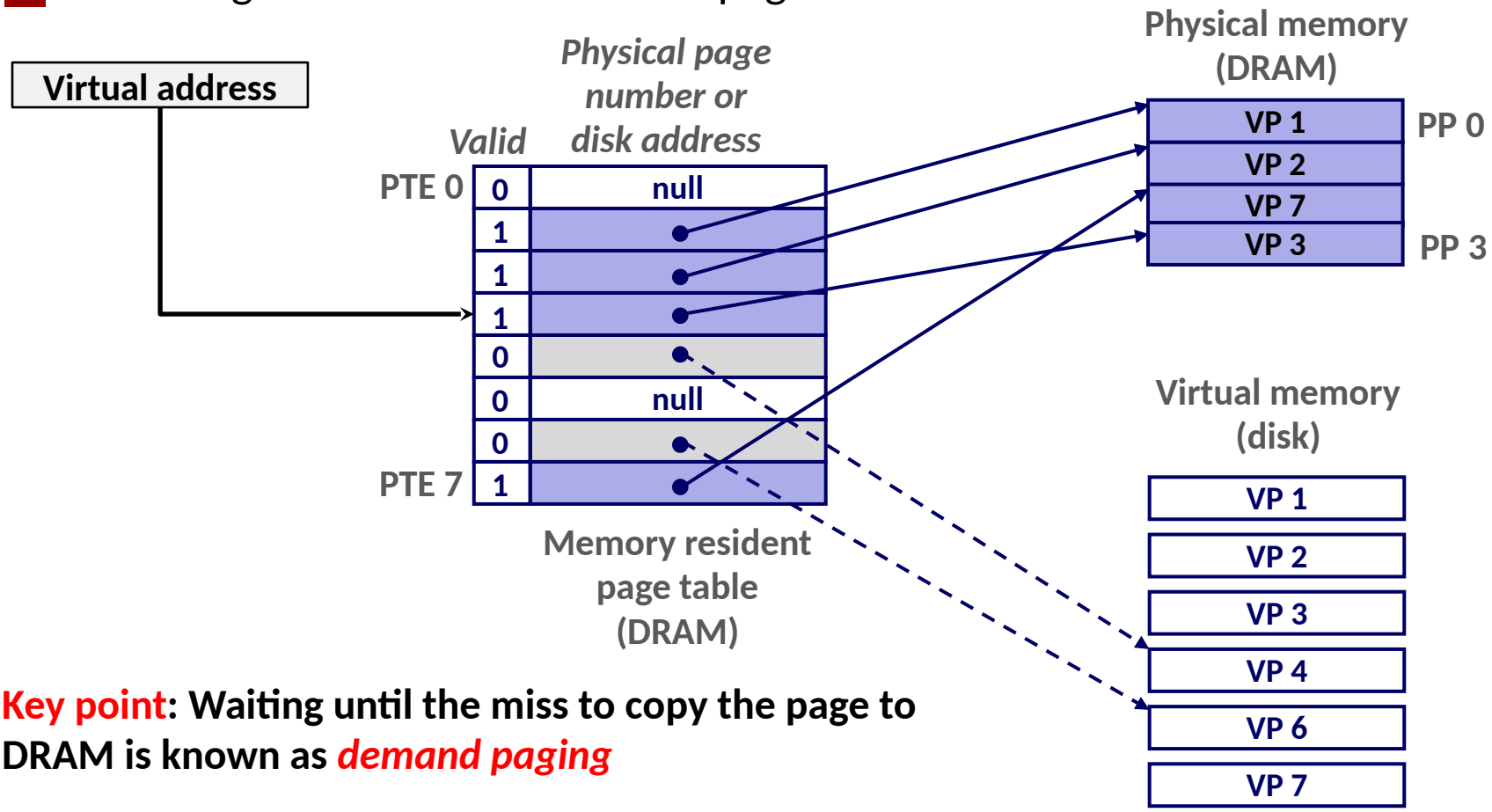
Handling Page Fault

- Page miss causes page fault (an exception)
- Operating system selects a victim to be evicted (here VP 4)



Handling Page Fault

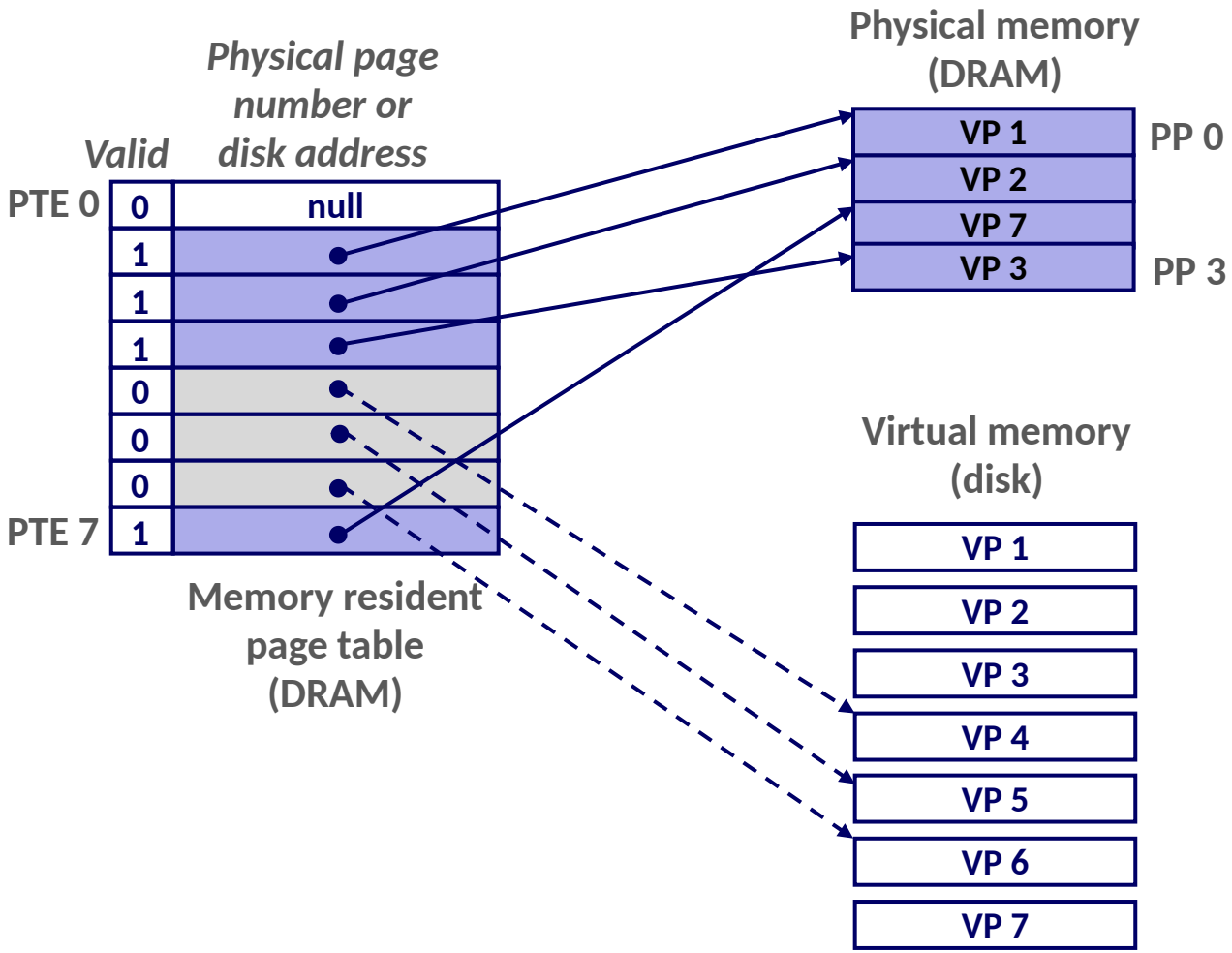
- Page miss causes page fault (an exception)
- Operating system selects a victim to be evicted (here VP 4)
- Offending instruction is restarted: page hit!



Key point: Waiting until the miss to copy the page to DRAM is known as *demand paging*

Allocating Pages

- Allocating a new page (VP 5) of virtual memory.



Locality to the Rescue Again!

- **Virtual memory seems terribly inefficient, but it works because of locality.**

- **At any point in time, programs tend to access a set of active virtual pages called the *working set***
  - Programs with better temporal locality will have smaller working sets

- **If (working set size < main memory size)**
  - Good performance for one process after compulsory misses

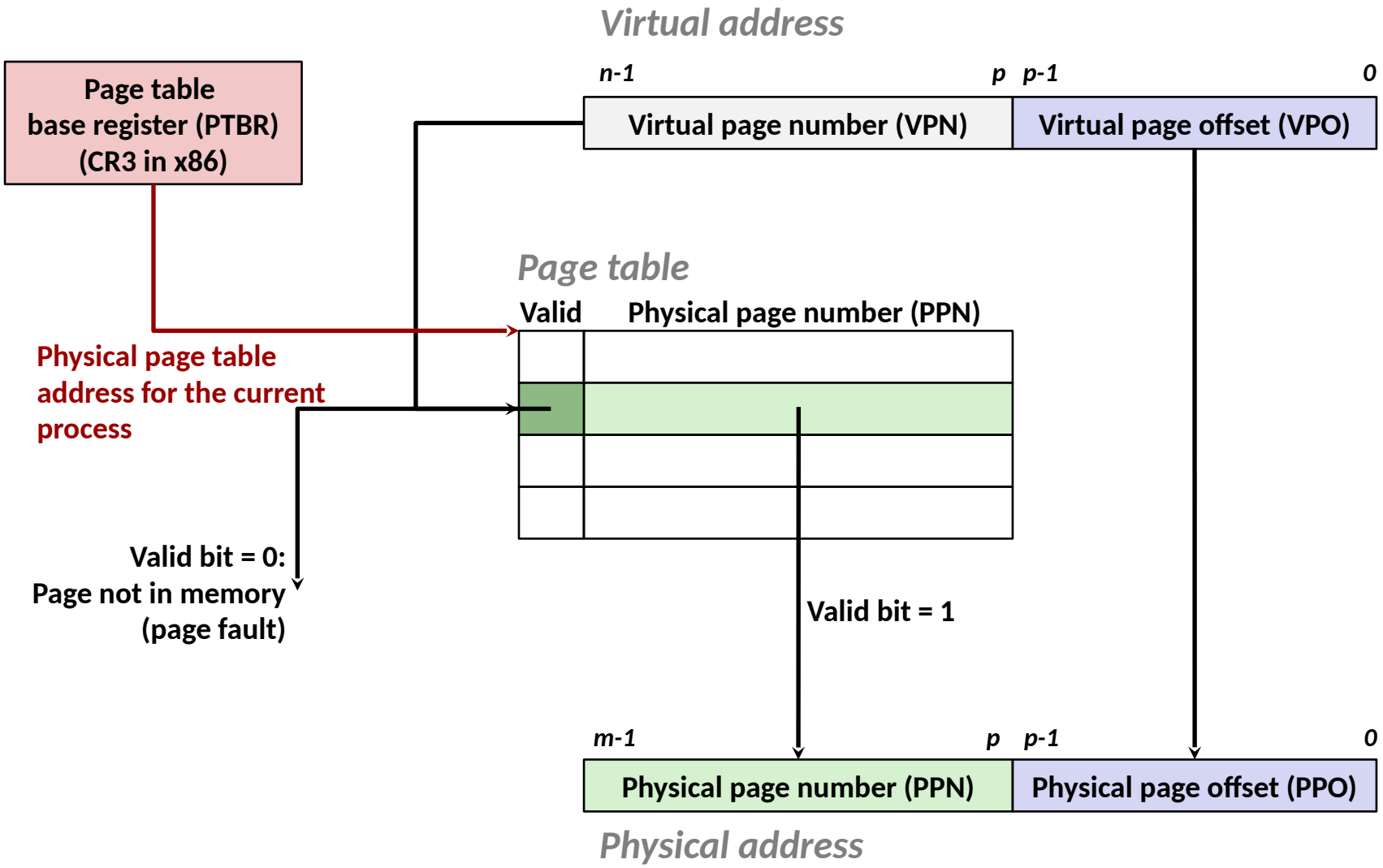
- **If (SUM(working set sizes) > main memory size)**
  - *Thrashing:* Performance meltdown where pages are swapped (copied) in and out continuously

Admission of Guilt

- **Lie: "Memory can be viewed as an array of bytes"…**
  - Actually discontinuous, with unmapped regions
- **Lie: "Memory addresses refer to locations in RAM"…**
  - Programmer sees only *virtual* addresses, which CPU's MMU translates to *physical* addresses before sending them to the memory controller
- **Lie: "Memory addresses are 64 bits"…**
  - Current x86-64 CPUs use 48-bit memory address buses, which is enough to address 256 TB of RAM
  - Future CPUs may widen this without a change to the ISA

Activity: part 4

Preview: Address Translation

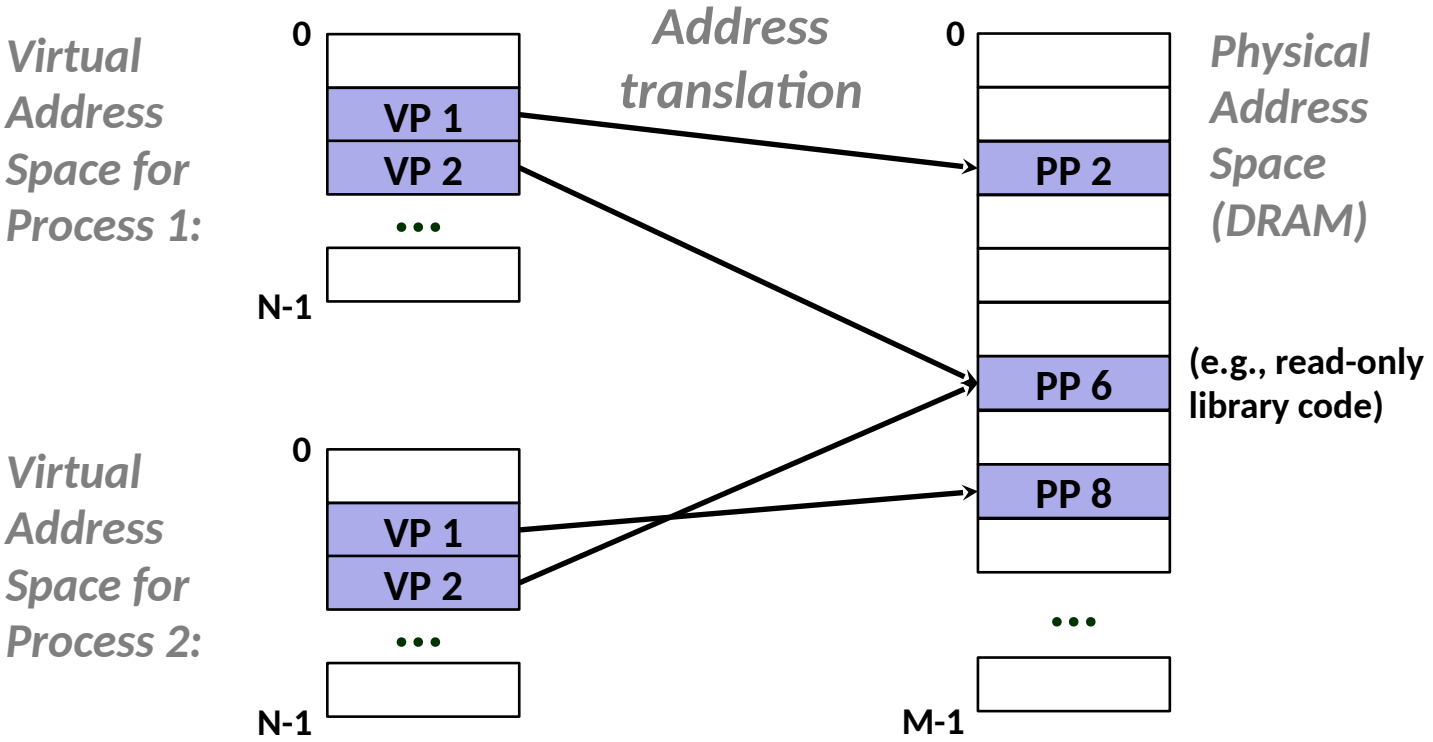


Today

- Processes: Concepts
- Address spaces
- VM as a tool for caching
- **VM as a tool for memory management**
- VM as a tool for memory protection

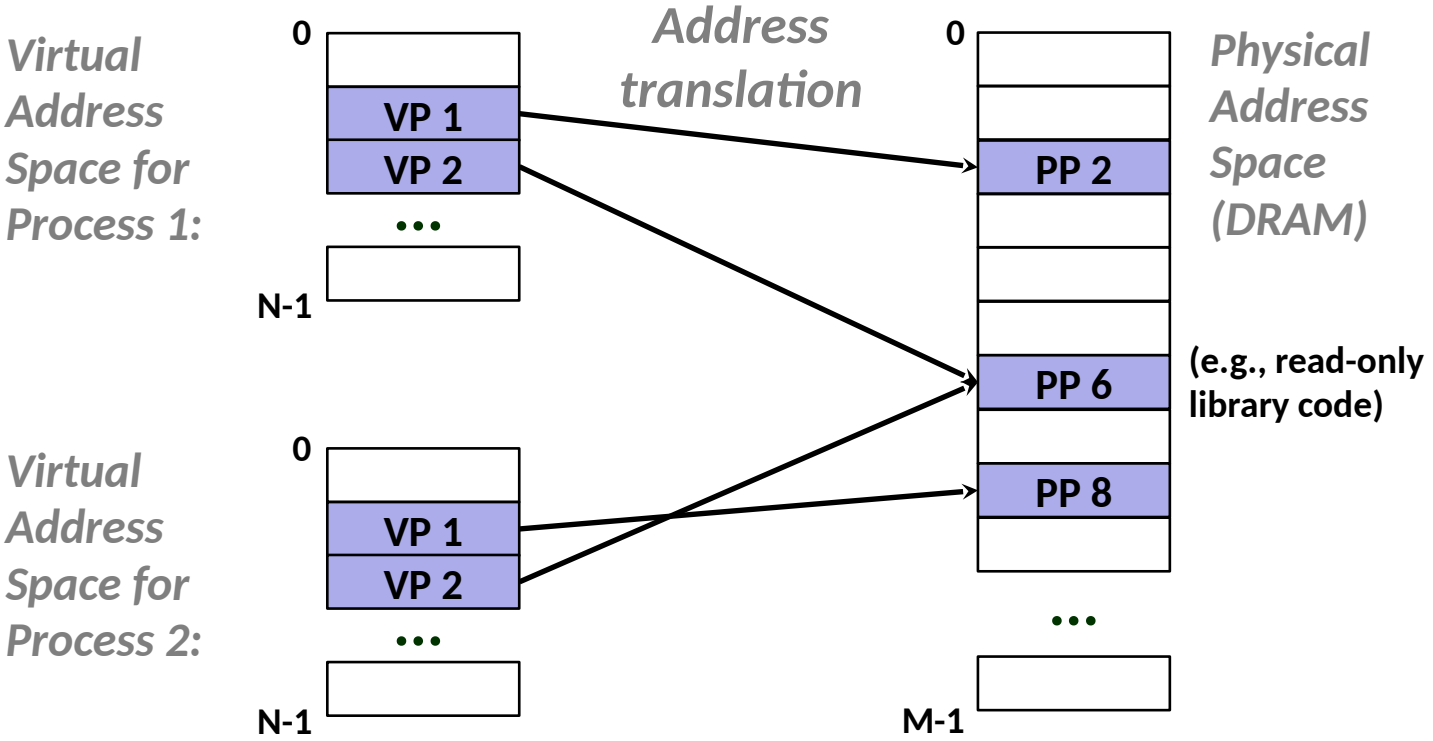
VM as a Tool for Memory Management

- **Key idea: each process has its own virtual address space**
  - It can view memory as a simple linear array
  - Mapping function scatters addresses through physical memory
    - Well-chosen mappings can improve locality



VM as a Tool for Memory Management

- **Simplifying memory allocation**
  - Each virtual page can be mapped to any physical page
  - A virtual page can be stored in different physical pages at different times
- **Sharing code and data among processes**
  - Map virtual pages to the same physical page (here: PP 6)



Simplifying Linking and Loading

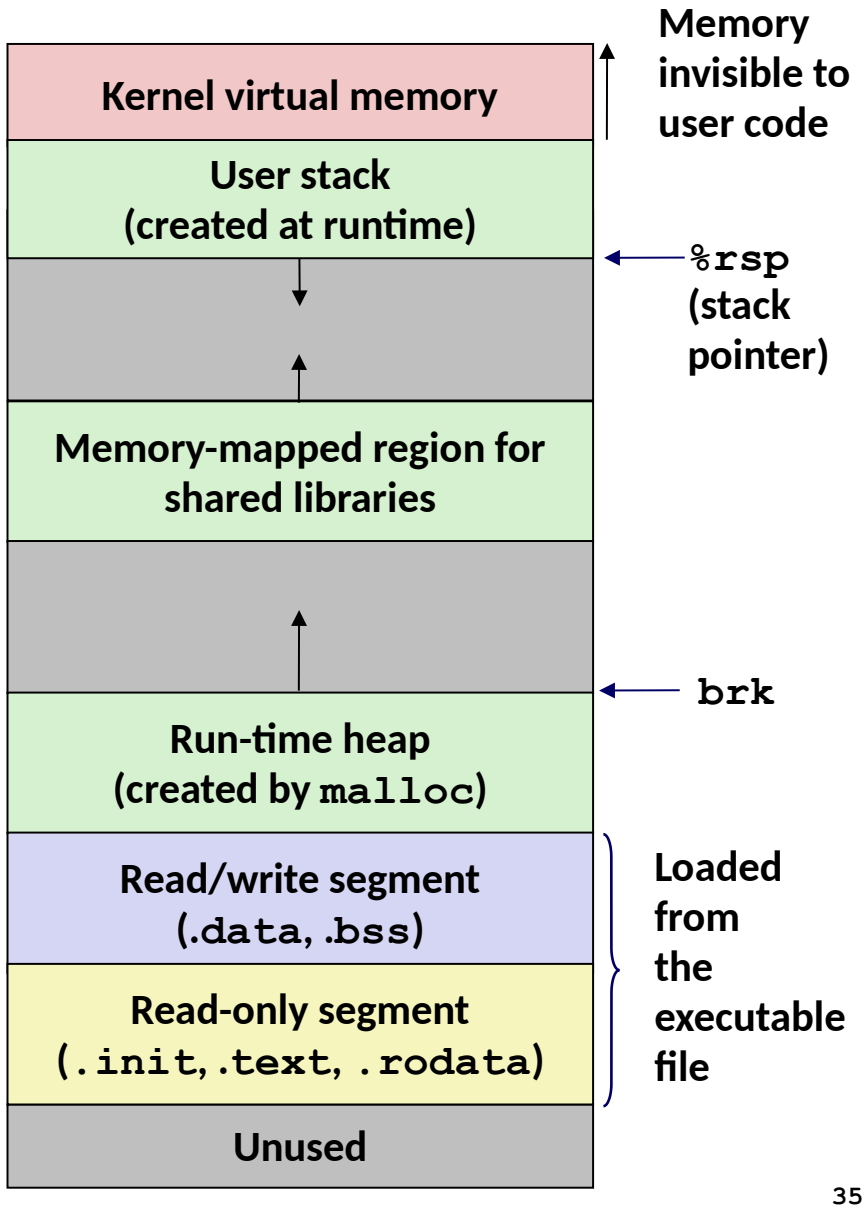
Linking

- Each program has similar virtual address space
- Code, data, and heap always start at the same addresses.

Loading

- Allocate virtual pages for `.text` and `.data` sections & creates PTEs marked as invalid
- The `.text` and `.data` sections are copied, page by page, on demand by the virtual memory system

Activity: parts 5 and 6 and quiz

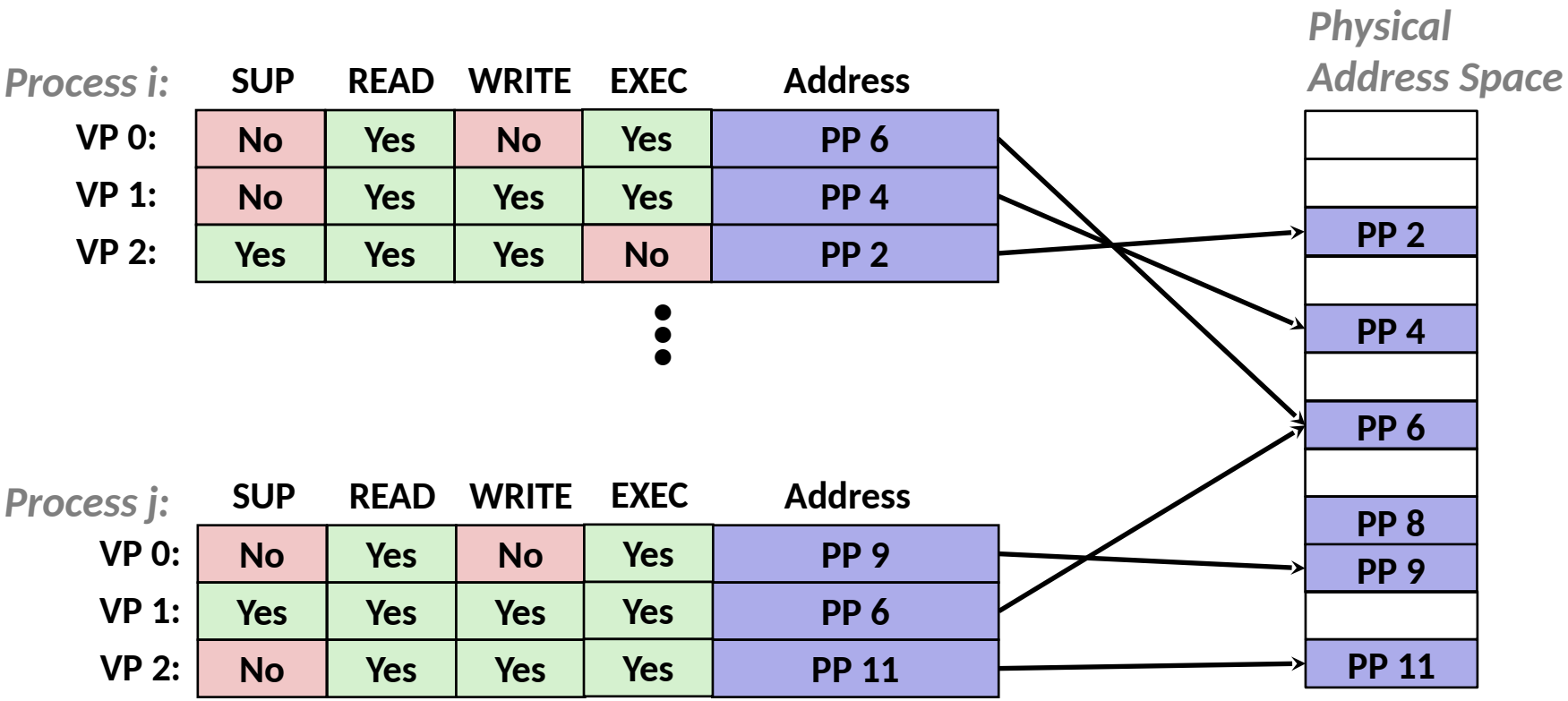


Today

- Processes: Concepts
- Address spaces
- VM as a tool for caching
- VM as a tool for memory management
- **VM as a tool for memory protection**

VM as a Tool for Memory Protection

- Extend PTEs with permission bits
- MMU checks these bits on each access

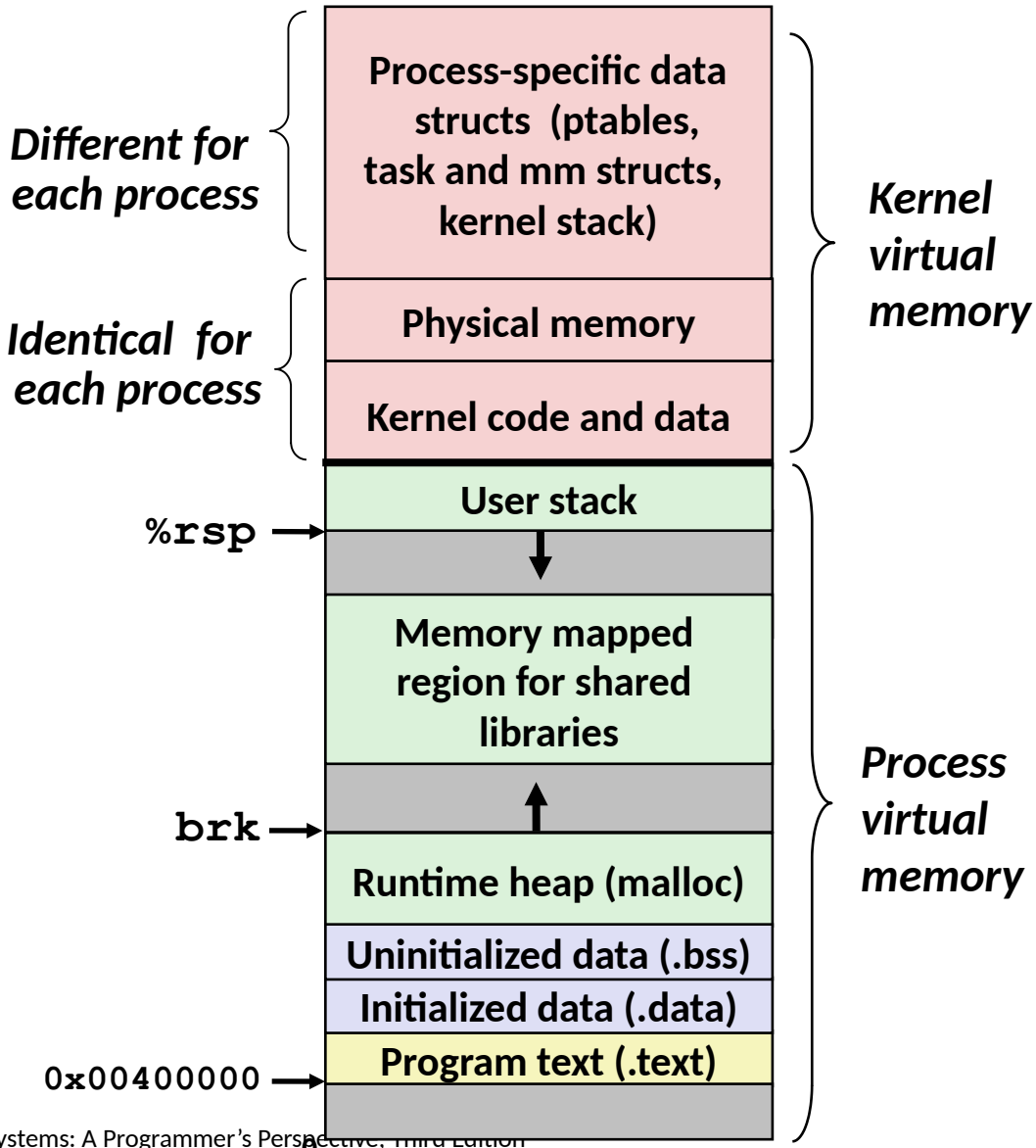


Summary

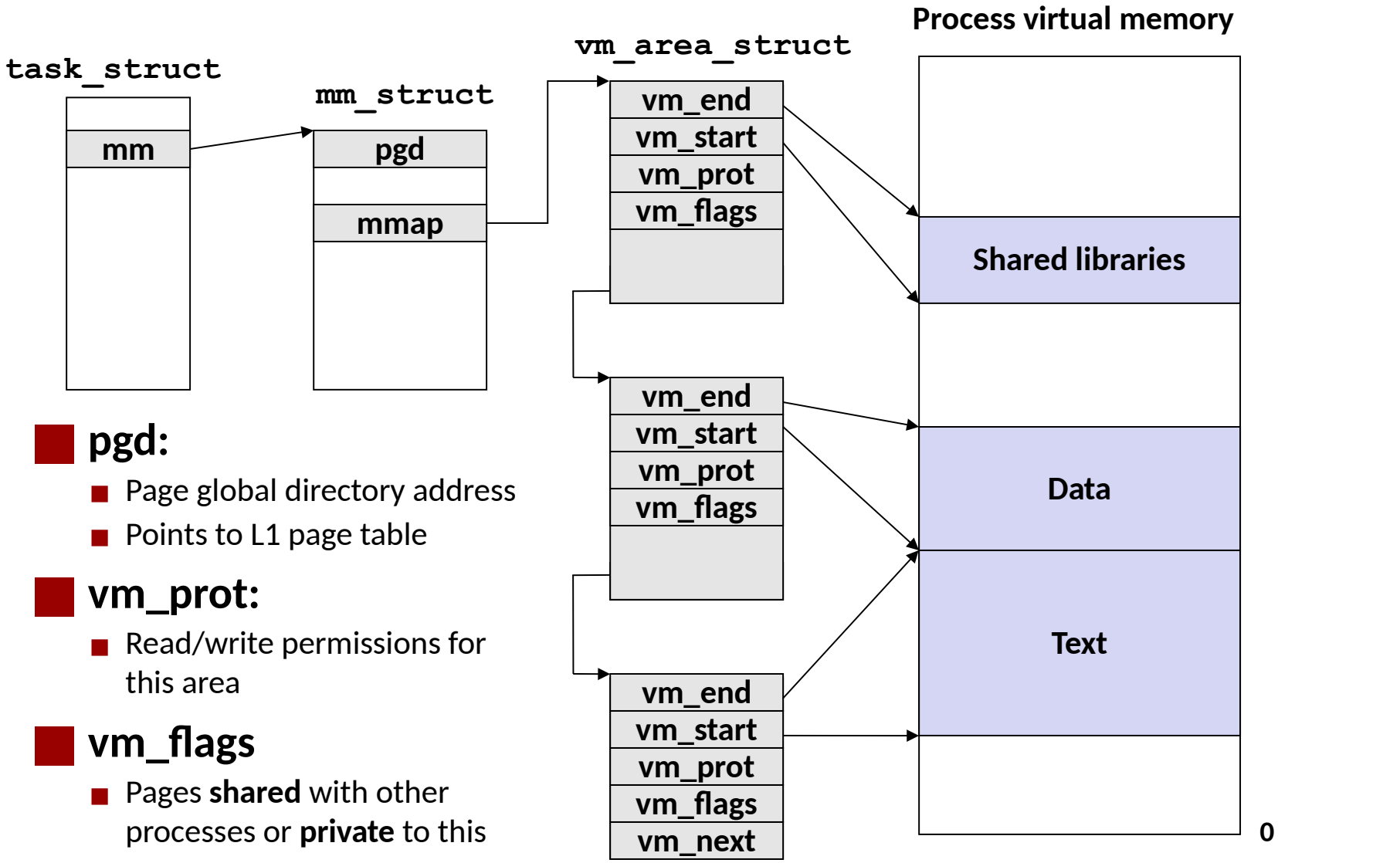
- **Programmer's view of virtual memory**
  - Each process has its own private linear address space
  - Cannot be corrupted by other processes

- **System view of virtual memory**
  - Uses memory efficiently by caching virtual memory pages
    - Efficient only because of locality
  - Simplifies memory management and programming
  - Simplifies protection by providing a convenient interpositioning point to check permissions

Virtual Address Space of a Linux Process



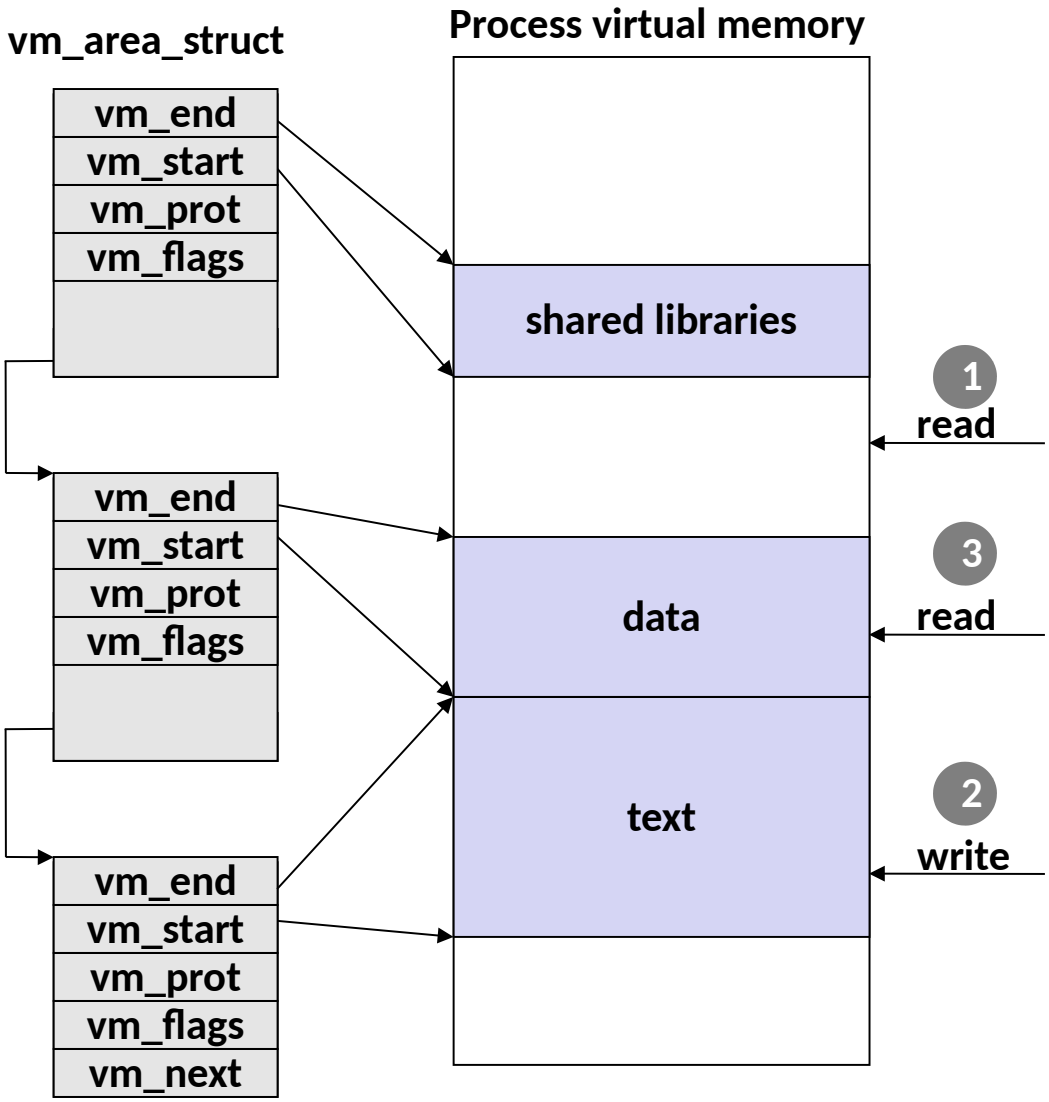
Linux Organizes VM as Collection of "Areas"



- **pgd:**
  - Page global directory address
  - Points to L1 page table
- **vm_prot:**
  - Read/write permissions for this area
- **vm_flags**
  - Pages **shared** with other processes or **private** to this process

Each process has own `task_struct`, etc

Linux Page Fault Handling



Segmentation fault:
accessing a non-existing page

Normal page fault

Protection exception:
e.g., violating permission by writing to a read-only page (Linux reports as Segmentation fault)